Abstract

The problem of solution transfer between meshes arises frequently in computational physics, e.g. in Lagrangian methods where remeshing occurs. The interpolation process must be conservative, i.e. it must conserve physical properties, such as mass. We extend previous works — which described the solution transfer process for straight sided unstructured meshes — by considering high-order isoparametric meshes with curved elements. The implementation is highly reliant on accurate computational geometry routines for evaluating points on and intersecting Bézier curves and triangles.

Keywords: Remapping, Curved Meshes, Lagrangian, Solution Transfer, Numerical analysis

Contents

1 Introduction

The first part is a general-purpose tool for computational physics problems. The tool enables solution transfer across two curved meshes. Since the tool requires a significant amount of computational geometry, the second half focuses on computational geometry. In particular, it considers cases where the geometric methods used have seriously degraded accuracy due to ill-conditioning.

In computational physics, the problem of solution transfer between meshes occurs in several applications. For example, by allowing the underyling computational domain to change during a simulation, computational effort can be focused dynamically to resolve sensitive features of a numerical solution. Mesh adaptivity

(see, for example, [BR78, PVMZ87, PUdOG01]), this in-flight change in the mesh, requires translating the numerical solution from the old mesh to the new, i.e. solution transfer. As another example, Lagrangian or particle-based methods treat each node in the mesh as a particle and so with each timestep the mesh travels *with* the fluid (see, for example, [HAC74]). However, over (typically limited) time the mesh becomes distorted and suffers a loss in element quality which causes catastrophic loss in the accuracy of computation. To overcome this, the domain must be remeshed or rezoned and the solution must be transferred (remapped) onto the new mesh configuration.

When pointwise interpolation is used to transfer a solution, quantities with physical meaning (e.g. mass, concentration, energy) may not be conserved. To address this, there have been many explorations (for example, [JH04, FPP+09, FM11]) of *conservative interpolation* (typically using Galerkin or L_2-minimizing methods). In this work, the author introduces a conservative interpolation method for solution transfer between high-order meshes. These high-order meshes are typically curved, but not necessarily all elements or at all timesteps.

The existing work on solution transfer has considered straight sided meshes, which use shape functions that have degree $p = 1$ to represent solutions on each element or so-called superparametric elements (i.e. a linear mesh with degree $p > 1$ shape functions on a regular grid of points). However, both to allow for greater geometric flexibility and for high order of convergence, this work will consider the case of curved isoparametric[1] meshes. Allowing curved geometries is useful since many practical problems involve geometries that change over time, such as flapping flight or fluid-structure interactions. In addition, high-order CFD methods ([WFA+13]) have the ability to produce highly accurate solutions with low dissipation and low dispersion error.

1.1 Actual Intro

In this chapter, an algorithm for conservative solution transfer between curved meshes will be described. This has practical applications to many methods in computational physics. Solution transfer is needed when a solution (approximated by a discrete field) is known on a *donor* mesh and must be transferred to a *target* mesh. In many applications, the field must be conserved for physical reasons, e.g. mass or energy cannot leave or enter the system, hence the focus on *conservative* solution transfer. A few scenarios where solution transfer is necessary will be considered below to motivate the "black box" solution transfer algorithm.

Since solution transfer is so commonly needed in physical applications, this problem of conservative interpolation has been considered already for straight sided meshes. The *common refinement* approach in [JH04] is used to compare several methods for solution transfer across two meshes. However, the problem of constructing a common refinement is not discussed there. The problem of constructing such a refinement is considered in [FPP+09, FM11] (called a supermesh by the authors). However, the solution transfer becomes considerably more challenging for curved meshes. For a sense of the difference between the straight sided and curved cases, consider the problem of intersecting an element from the donor mesh with an element from the target mesh. If the elements are triangles, the intersection is either a convex polygon or has measure zero. If the elements are curved, the intersection can be non-convex and can even split into multiple disjoint regions.

1.1.1 Lagrangian Methods

The method of characteristics helps transform partial differential equations into ordinary differential equations by dividing the physical domain into a family of curves. For example, the simple transport equation

$$u_t + cu_x = 0 \tag{1}$$

can be transformed when restricting to the family of lines $x(t) = x_0 + ct$. On these lines $u(x(t), t)$ is constant, by construction, and so the solution is "transported" from $u(x_0, 0)$ along each characteristic line (Figure 1).

Motivated by this, *Lagrangian methods* treat each point in the physical domain as a "particle" which moves along a characteristic curve over time and then monitor values associated with the particle (heat /

[1] I.e. the degree of the discrete field on the mesh is same as the degree of the shape functions that determine the mesh.

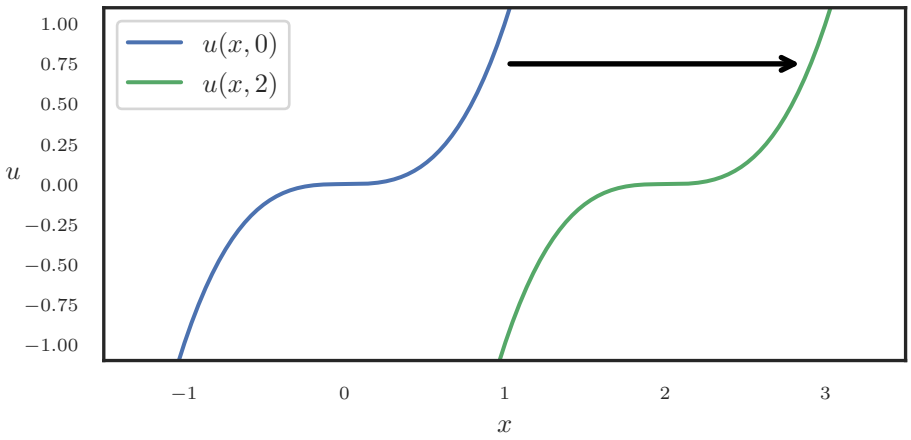

Figure 1: The solution to $u_t + u_x = 0,\ u(x,0) = x^3$ plotted in the xu-plane. Demonstrates simple transport of the solution.

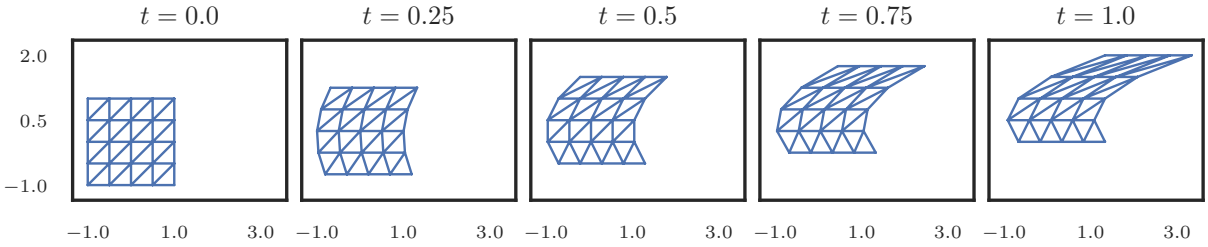

Figure 2: Distortion of a regular mesh caused by particle motion along the velocity field $\begin{bmatrix} y^2 & 1 \end{bmatrix}^T$ from $t = 0$ to $t = 1$ with $\Delta t = 1/4$.

energy, velocity, pressure, density, concentration, etc.). They are an effective way to solve PDEs, even with higher order or non-linear terms. For example, if a diffusion term is added to (1)

$$u_t + cu_x - \varepsilon u_{xx} = 0 \tag{2}$$

then the same characteristics can be used, but the value along each characteristic is no longer constant; instead it satisfies the ODE $\frac{d}{dt}u(x(t),t) = \varepsilon u_{xx}$.

This approach transforms the numerical solution of PDEs into a family of numerical solutions to many independent ODEs. It allows the use of familiar and well understood ODE solvers. In addition, Lagrangian methods often have less restrictive conditions on time steps than Eulerian methods[2]. When solving PDEs on unstructured meshes with Lagrangian methods, the nodes move (since they are treated like particles) and the mesh "travels".

1.1.2 Remeshing and Adaptivity

A flow-based change to a mesh can cause problems if it causes the mesh to leave the domain being analyzed or if it distorts the mesh until the element quality is too low in some mesh elements. Over enough time, the mesh can even tangle (i.e. elements begin to overlap). For an example of such distortion (Figure 2), consider a PDE of the form

$$u_t + \begin{bmatrix} y^2 \\ 1 \end{bmatrix} \cdot \nabla u + F\left(u, \nabla u\right) = 0. \tag{3}$$

[2]In Eulerian methods, the mesh is fixed.

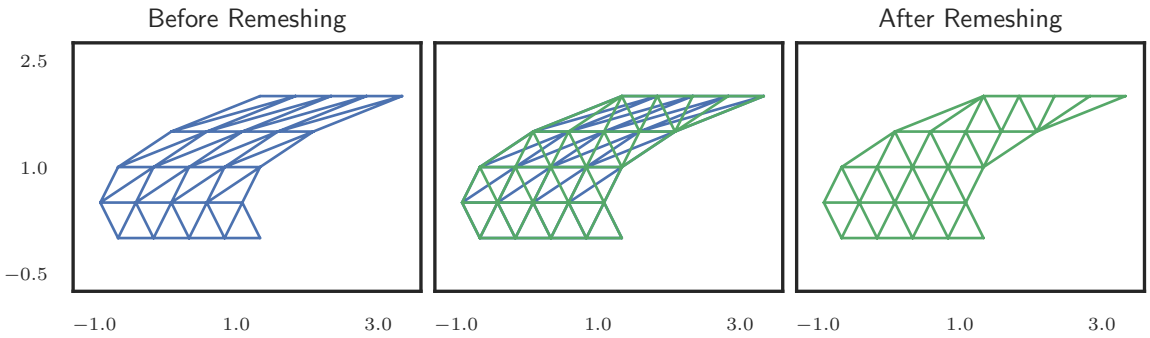

Figure 3: Remeshing a domain after distortion caused by particle motion along the velocity field $\begin{bmatrix} y^2 & 1 \end{bmatrix}^T$ from $t = 0$ to $t = 1$.

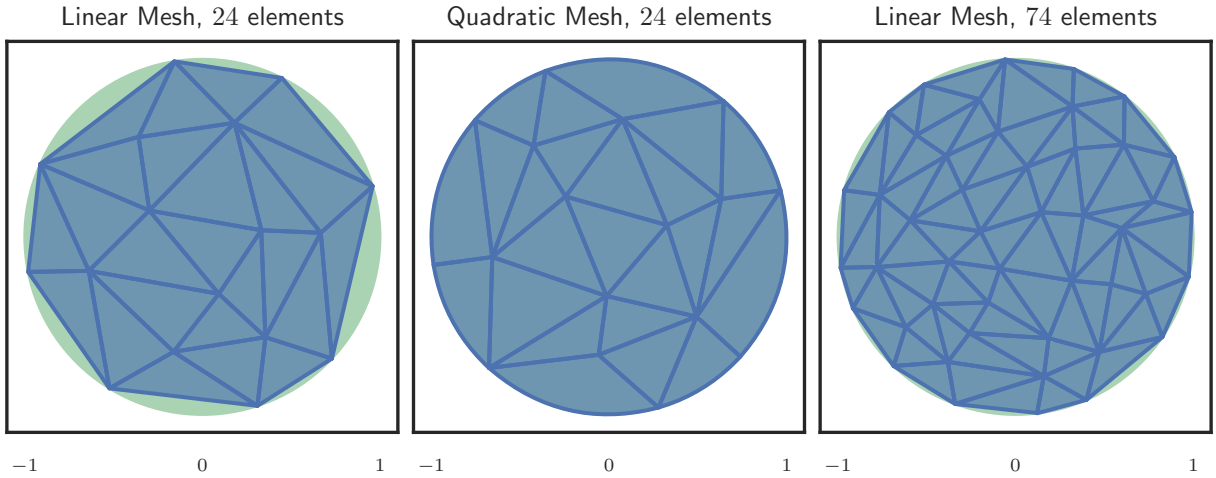

Figure 4: Comparing straight sided meshes to a curved mesh when approximating the unit disc in $\mathbf{R}^2$.

The characteristics $y(t) = y_0 + t, x(t) = x_0 + \left(y(t)^3 - y_0^3\right)/3$ distort the mesh considerably after just one second.

To deal with distortion, one can allow the mesh to adapt in between time steps. For example, Figure 3 shows an example remeshing of the domain. In addition to improving mesh quality, mesh adaptivity can be used to dynamically focus computational effort to resolve sensitive features of a numerical solution. From [IK04]

> In order to balance the method's approximation quality and its computational costs effectively, adaptivity is an essential requirement, especially when modelling multiscale phenomena.

For more on mesh adaptivity, see [BR78, PVMZ87, PUdOG01].

In either case, the change in the mesh between time steps requires transferring a known solution on the discarded mesh to the mesh produced by the remeshing process. Without the ability to change the mesh, Lagrangian methods (or, more generally, ALE [HAC74]) would not be useful, since after a limited time the mesh will distort.

1.1.3 High-order Meshes

To allow for greater geometric flexibility and for high order of convergence, curved mesh elements can be used in the finite element method. Though the complexity of a method can steeply rise when allowing curved elements, the trade for high-order convergence can be worth it. (See [WFA+13] for more on high-order CFD

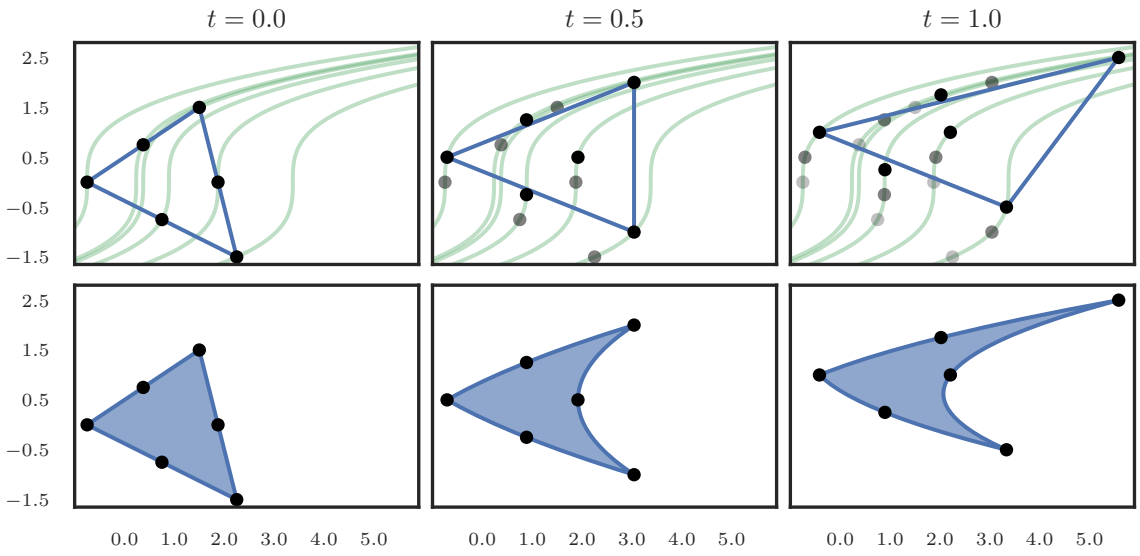

Figure 5: Movement of nodes in a quadratic element under distortion caused by particle motion along the velocity field $\begin{bmatrix} y^2 & 1 \end{bmatrix}^T$ from $t = 0$ to $t = 1$ with $\Delta t = 1/2$. The green curves represent the characteristics that each node travels along.

methods.) Curved meshes can typically represent a given geometry with far fewer elements than a straight sided mesh (for example, Figure 4). The increase in accuracy also allows for the use of fewer elements, which in turn can also facilitate a reduction in the overall computation time.

Even if the domain has no inherent curvature, high-order (degree p) shape functions allow for order $p+1$ convergence, which is desirable in it's own right. However, even in such cases, a Lagrangian method must either curve the mesh or information about the flow of the geometry will be lost. Figure 5 shows what happens to a given quadratic element as the nodes move along the characteristics from (3). This element uses the triangle vertices and edge midpoints to determine the shape functions. However, as the nodes move with the flow, the midpoints are no longer on the lines connecting the vertex nodes. To allow the mesh to more accurately represent the solution, the edges can instead curve so that the midpoint nodes remain halfway between (i.e. half of the parameter space) the vertex nodes along an edge.

1.1.4 Multiphysics and Comparing Methods

In multiphysics simulations, a problem is partitioned into physical components. This partitioning can apply to both the physical domain (e.g. separating a solid and fluid at an interface) and the simulation solution itself (e.g. solving for pressure on one mesh and velocity on another). Each (multi)physics component is solved for on its own mesh. When the components interact, the simulation solution must be transferred between those meshes.

In a similar category of application, solution transfer enables the comparison of solutions defined on different meshes. For example, if a reference solution is known on a very fine special-purpose mesh, the error can be computed for a coarse mesh by transferring the solution from the fine mesh and taking the difference. Or, if the same method is used on different meshes of the same domain, the resulting computed solutions can be compared via solution transfer. Or, if two different methods use two different meshes of the same domain.

1.1.5 Local versus Global Transfer

Conservative solution transfer has been around since the advent of ALE, and as a result much of the existing literature focuses on mesh-mesh pairs that will occur during an ALE-based simulation. When flow-based mesh distortion occurs, elements are typically "flipped" (e.g. a diagonal is switched in a pair of elements) or

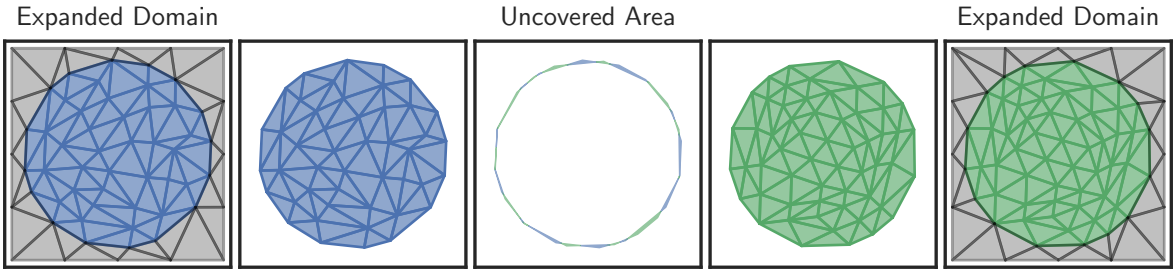

Figure 6: Partially overlapping meshes on a near identical domain. Both are linear meshes that approximate the unit disc in $\mathbf{R}^2$. The outermost columns show how the domain of each mesh can be expanded so they agree.

elements are subdivided or combined. These operations are inherently local, hence the solution transfer can be done locally across known neighbors. Typically, this locality is crucial to solution transfer methods. In [MS03], the transfer is based on partitioning elements of the updated mesh into components of elements from the old mesh and "swept regions" from neighbouring elements. In [KS08], the (locally) changing connectivity of the mesh is addressed. In [GKS07], the local transfer is done on polyhedral meshes.

Global solution transfer instead seeks to conserve the solution across the whole mesh. It makes no assumptions about the relationship between the donor and target meshes. The loss in local information makes the mesh intersection problem more computationally expensive, but the added flexibility reduces timestep restrictions since it allows remeshing to be done less often. In [Duk84, DK87], a global transfer is enabled by transforming volume integrals to surface integrals via the divergence theorem to reduce the complexity of the problem.

1.1.6 Limitations

The method described in this work only applies to meshes in $\mathbf{R}^2$. Application to meshes in $\mathbf{R}^3$ is a direction for future research, though the geometric kernels (see Chapter 3) become significantly more challenging to describe and implement. In addition, the method will assume that every element in the target mesh is contained in the donor mesh. This ensures that the solution transfer is *interpolation*. In the case where all target elements are partially covered, *extrapolation* could be used to extend a solution outside the domain, but for totally uncovered elements there is no clear correspondence to elements in the donor mesh.

The case of partially overlapping meshes can be addressed in particular cases (i.e. with more information). For example, consider a problem defined on $\Omega = \mathbf{R}^2$ and solution that tends towards zero as points tend to infinity. A typical approach may be to compute the solution on a circle of large enough radius and consider the numerical solution to be zero outside the circle. Figure 6 shows how solution transfer could be performed in such cases when the meshes partially overlap: construct a simple region containing both computational domains and then mesh the newly introduced area. However, the assumption that the numerical solution is zero in the newly introduced area is very specific and a similar approach may not apply in other cases of partial overlap.

Some attempts ([Ber87, CH94, CDS99]) have been made to interpolate fluxes between overlapping meshes. These perform an interpolation on the region common to both meshes and then numerically solve the PDE to determine the values on the uncovered elements.

When solving some PDEs with viscoelastic fields, additional nodes must be tracked at quadrature points. As the mesh travels, these quadrature points must be moved as well in a way that is consistent with the standard nodes that define the element. The solution transfer method described here does not provide a way either to determine quadrature points on a target mesh or to map the field onto known quadrature points.

1.2 Overview

This work is organized as follows. Section 2 establishes common notation and reviews basic results relevant to the topics at hand. Section 3 is an in-depth discussion of the computational geometry methods needed

to implement to enable solution transfer. Section 4 describes the solution transfer process and gives results of some numerical experiments confirming the rate of convergence.

2 Preliminaries

2.1 General Notation

We'll refer to $\mathbf{R}$ for the reals, $\mathcal{U}$ represents the unit triangle (or unit simplex) in $\mathbf{R}^2$: $\mathcal{U} = \{(s,t) \mid 0 \leq s, t, s+t \leq 1\}$. When dealing with sequences with multiple indices, e.g. $s_{m,n} = m+n$, we'll use bold symbols to represent a multi-index: $\boldsymbol{i} = (m,n)$. We'll use $|\boldsymbol{i}|$ to represent the sum of the components in a multi-index. The binomial coefficient $\binom{n}{k}$ is equal to $\frac{n!}{k!(n-k)!}$ and the trinomial coefficient $\binom{n}{i,j,k}$ is equal to $\frac{n!}{i!j!k!}$ (where $i+j+k=n$). The notation δ_{ij} represents the Kronecker delta, a value which is 1 when $i=j$ and 0 otherwise.

2.2 Bézier Curves

A *Bézier curve* is a mapping from the unit interval that is determined by a set of control points $\{\boldsymbol{p}_j\}_{j=0}^n \subset \mathbf{R}^d$. For a parameter $s \in [0,1]$, there is a corresponding point on the curve:

$$b(s) = \sum_{j=0}^{n} \binom{n}{j} (1-s)^{n-j} s^j \boldsymbol{p}_j \in \mathbf{R}^d. \tag{4}$$

This is a combination of the control points weighted by each Bernstein basis function $B_{j,n}(s) = \binom{n}{j}(1-s)^{n-j}s^j$. Due to the binomial expansion $1 = (s + (1-s))^n = \sum_{j=0}^n B_{j,n}(s)$, a Bernstein basis function is in $[0,1]$ when s is as well. Due to this fact, the curve must be contained in the convex hull of it's control points.

2.3 Bézier Triangles

A *Bézier triangle* ([Far01, Chapter 17]) is a mapping from the unit triangle $\mathcal{U}$ and is determined by a control net $\{\boldsymbol{p}_{i,j,k}\}_{i+j+k=n} \subset \mathbf{R}^d$. A Bézier triangle is a particular kind of Bézier surface, i.e. one in which there are two cartesian or three barycentric input parameters. Often the term Bézier surface is used to refer to a tensor product or rectangular patch. For $(s,t) \in \mathcal{U}$ we can define barycentric weights $\lambda_1 = 1-s-t, \lambda_2 = s, \lambda_3 = t$ so that

$$1 = (\lambda_1 + \lambda_2 + \lambda_3)^n = \sum_{\substack{i+j+k=n \\ i,j,k \geq 0}} \binom{n}{i,j,k} \lambda_1^i \lambda_2^j \lambda_3^k. \tag{5}$$

Using this we can similarly define a (triangular) Bernstein basis

$$B_{i,j,k}(s,t) = \binom{n}{i,j,k} (1-s-t)^i s^j t^k = \binom{n}{i,j,k} \lambda_1^i \lambda_2^j \lambda_3^k \tag{6}$$

that is in $[0,1]$ when (s,t) is in $\mathcal{U}$. Using this, we define points on the Bézier triangle as a convex combination of the control net:

$$b(s,t) = \sum_{i+j+k=n} \binom{n}{i,j,k} \lambda_1^i \lambda_2^j \lambda_3^k \boldsymbol{p}_{i,j,k} \in \mathbf{R}^d. \tag{7}$$

Rather than defining a Bézier triangle by the control net, it can also be uniquely determined by the image of a standard lattice of points in $\mathcal{U}$: $b\left(j/n, k/n\right) = \boldsymbol{n}_{i,j,k}$; we'll refer to these as *standard nodes*. Figure 7 shows these standard nodes for a cubic triangle in $\mathbf{R}^2$. To see the correspondence, when $p=1$ the standard nodes *are* the control net

$$b(s,t) = \lambda_1 \boldsymbol{n}_{1,0,0} + \lambda_2 \boldsymbol{n}_{0,1,0} + \lambda_3 \boldsymbol{n}_{0,0,1} \tag{8}$$

and when $p=2$

$$b(s,t) = \lambda_1 \left(2\lambda_1 - 1\right) \boldsymbol{n}_{2,0,0} + \lambda_2 \left(2\lambda_2 - 1\right) \boldsymbol{n}_{0,2,0} + \lambda_3 \left(2\lambda_3 - 1\right) \boldsymbol{n}_{0,0,2} +$$

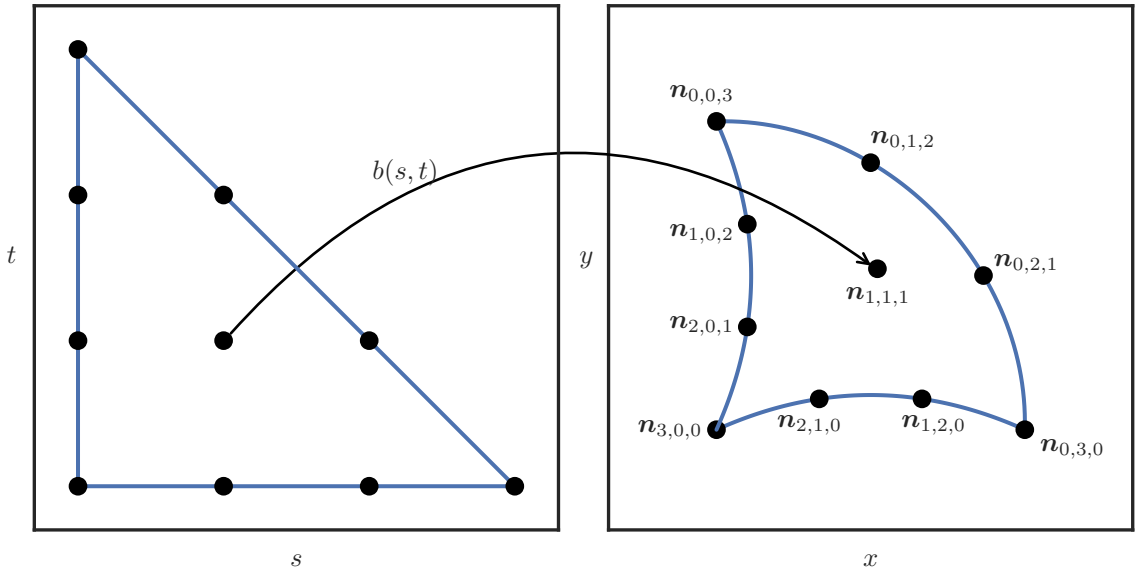

Figure 7: Cubic Bézier triangle

$$4\lambda_1\lambda_2\boldsymbol{n}_{1,1,0} + 4\lambda_2\lambda_3\boldsymbol{n}_{0,1,1} + 4\lambda_3\lambda_1\boldsymbol{n}_{1,0,1}. \quad (9)$$

However, it's worth noting that the transformation between the control net and the standard nodes has condition number that grows exponentially with n (see [Far91], which is related but does not directly show this). This may make working with higher degree triangles prohibitively unstable.

A *valid* Bézier triangle is one which is diffeomorphic to $\mathcal{U}$, i.e. $b(s,t)$ is bijective and has an everywhere invertible Jacobian. We must also have the orientation preserved, i.e. the Jacobian must have positive determinant. For example, in Figure 8, the image of $\mathcal{U}$ under the map $b(s,t) = \begin{bmatrix} (1-s-t)^2 + s^2 & s^2 + t^2 \end{bmatrix}^T$ is not valid because the Jacobian is zero along the curve $s^2 - st - t^2 - s + t = 0$ (the dashed line). Elements that are not valid are called *inverted* because they have regions with "negative area". For the example, the image $b(\mathcal{U})$ leaves the boundary determined by the edge curves: $b(r,0)$, $b(1-r,r)$ and $b(0,1-r)$ when $r \in [0,1]$. This region outside the boundary is traced twice, once with a positive Jacobian and once with a negative Jacobian.

2.4 Curved Elements

We define a curved mesh element $\mathcal{T}$ of degree p to be a Bézier triangle in $\mathbf{R}^2$ of the same degree. We refer to the component functions of $b(s,t)$ (the map that gives $\mathcal{T} = b(\mathcal{U})$) as $x(s,t)$ and $y(s,t)$.

This fits a typical definition ([JM09, Chapter 12]) of a curved element, but gives a special meaning to the mapping from the reference triangle. Interpreting elements as Bézier triangles has been used for Lagrangian methods where mesh adaptivity is needed (e.g. [CMOP04]). Typically curved elements only have one curved side ([MM72]) since they are used to resolve geometric features of a boundary. See also [Zlá73, Zlá74]. Bézier curves and triangles have a number of mathematical properties (e.g. the convex hull property) that lead to elegant geometric descriptions and algorithms.

Note that a Bézier triangle can be determined from many different sources of data (for example the control net or the standard nodes). The choice of this data may be changed to suit the underlying physical problem without changing the actual mapping. Conversely, the data can be fixed (e.g. as the control net) to avoid costly basis conversion; once fixed, the equations of motion and other PDE terms can be recast relative to the new basis (for an example, see [PBP09], where the domain varies with time but the problem is reduced to solving a transformed conservation law in a fixed reference configuration).

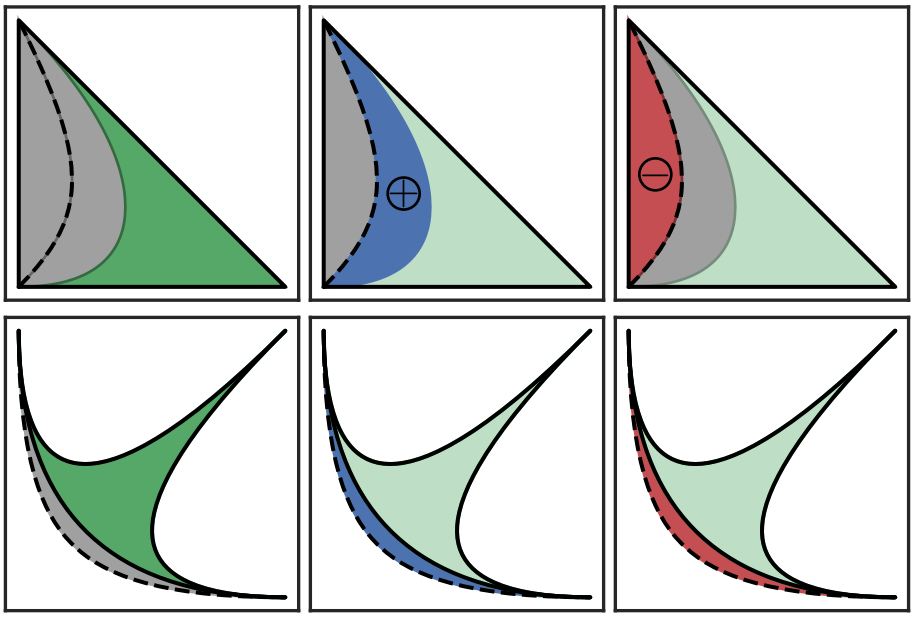

Figure 8: The Bézier triangle given by $b(s,t) = \left[(1-s-t)^2 + s^2 \;\; s^2 + t^2\right]^T$ produces an inverted element. It traces the same region twice, once with a positive Jacobian (the middle column) and once with a negative Jacobian (the right column).

2.5 Shape Functions

When defining shape functions (i.e. a basis with geometric meaning) on a curved element there are (at least) two choices. When the degree of the shape functions is the same as the degree of the function being represented on the Bézier triangle, we say the element $\mathcal{T}$ is *isoparametric*. For the multi-index $\boldsymbol{i} = (i, j, k)$, we define $\boldsymbol{u_i} = (j/n, k/n)$ and the corresponding standard node $\boldsymbol{n_i} = b\left(\boldsymbol{u_i}\right)$. Given these points, two choices for shape functions present themselves:

- *Pre-Image Basis*: $\phi_{\boldsymbol{j}}\left(\boldsymbol{n_i}\right) = \widehat{\phi}_{\boldsymbol{j}}\left(\boldsymbol{u_i}\right) = \widehat{\phi}_{\boldsymbol{j}}\left(b^{-1}\left(\boldsymbol{n_i}\right)\right)$ where $\widehat{\phi}_{\boldsymbol{j}}$ is a canonical basis function on $\mathcal{U}$, i.e. $\widehat{\phi}_{\boldsymbol{j}}$ a degree p bivariate polynomial and $\widehat{\phi}_{\boldsymbol{j}}\left(\boldsymbol{u_i}\right) = \delta_{\boldsymbol{ij}}$
- *Global Coordinates Basis*: $\phi_{\boldsymbol{j}}\left(\boldsymbol{n_i}\right) = \delta_{\boldsymbol{ij}}$, i.e. a canonical basis function on the standard nodes $\{\boldsymbol{n_i}\}$.

For example, consider a quadratic Bézier triangle:

$$b(s,t) = \begin{bmatrix} 4(st+s+t) & 4(st+t+1) \end{bmatrix}^T \tag{10}$$

$$\implies \begin{bmatrix} \boldsymbol{n}_{2,0,0} & \boldsymbol{n}_{1,1,0} & \boldsymbol{n}_{0,2,0} & \boldsymbol{n}_{1,0,1} & \boldsymbol{n}_{0,1,1} & \boldsymbol{n}_{0,0,2} \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 & 2 & 5 & 4 \\ 4 & 4 & 4 & 6 & 7 & 8 \end{bmatrix}. \tag{11}$$

In the *Global Coordinates Basis*, we have

$$\phi^G_{0,1,1}(x,y) = \frac{(y-4)(x-y+4)}{6}. \tag{12}$$

For the *Pre-Image Basis*, we need the inverse and the canonical basis

$$b^{-1}(x,y) = \begin{bmatrix} \frac{x-y+4}{4} & \frac{y-4}{x-y+8} \end{bmatrix} \quad \text{and} \quad \widehat{\phi}_{0,1,1}(s,t) = 4st \tag{13}$$

and together they give

$$\phi^P_{0,1,1}(x,y) = \frac{(y-4)(x-y+4)}{x-y+8}. \tag{14}$$

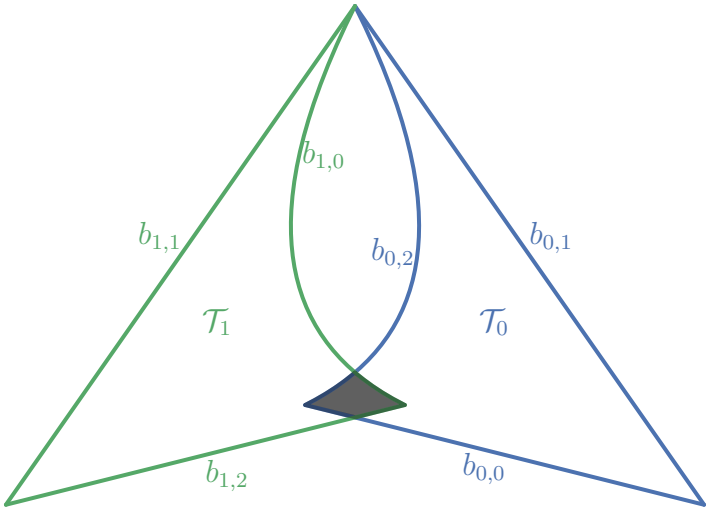

Figure 9: Intersection of Bézier triangles form a curved polygon.

In general ϕ_j^P may not even be a rational bivariate function; due to composition with b^{-1} we can only guarantee that it is algebraic (i.e. it can be defined as the zero set of polynomials).

2.6 Curved Polygons

When intersecting two curved elements, the resulting surface(s) will be defined by the boundary, alternating between edges of each element. For example, in Figure 9, a "curved quadrilateral" is formed when two Bézier triangles $\mathcal{T}_0$ and $\mathcal{T}_1$ are intersected.

A *curved polygon* is defined by a collection of Bézier curves in $\mathbf{R}^2$ that determine the boundary. In order to be a valid polygon, none of the boundary curves may cross, the ends of consecutive edge curves must meet and the curves must be right-hand oriented. For our example in Figure 9, the triangles have boundaries formed by three Bézier curves: $\partial\mathcal{T}_0 = b_{0,0} \cup b_{0,1} \cup b_{0,2}$ and $\partial\mathcal{T}_1 = b_{1,0} \cup b_{1,1} \cup b_{1,2}$. The intersection $\mathcal{P}$ is defined by four boundary curves: $\partial\mathcal{P} = C_1 \cup C_2 \cup C_3 \cup C_4$. Each boundary curve is itself a Bézier curve[3]: $C_1 = b_{0,0}([0, 1/8])$, $C_2 = b_{1,2}([7/8, 1])$, $C_3 = b_{1,0}([0, 1/7])$ and $C_4 = b_{0,2}([6/7, 1])$.

Though an intersection can be described in terms of the Bézier triangles, the structure of the control net will be lost. The region will not in general be able to be described by a mapping from a simple space like $\mathcal{U}$.

3 Bézier Intersection Problems

3.1 Intersecting Bézier Curves

The problem of intersecting two Bézier curves is a core building block for intersecting two Bézier triangles in $\mathbf{R}^2$ Since a curve is an algebraic variety of dimension one, the intersections will either be a curve segment common to both curves (if they coincide) or a finite set of points (i.e. dimension zero). Many algorithms have been described in the literature, both geometric ([SP86, SN90, KLS98]) and algebraic ([MD92]).

In the implementation for this work, the Bézier subdivision algorithm is used. In the case of a transversal intersection (i.e. one where the tangents to each curve are not parallel and both are non-zero), this algorithm performs very well. However, when curves are tangent, a large number of (false) candidate intersections are detected and convergence of Newton's method slows once in a neighborhood of an actual intersection. Non-transversal intersections have infinite condition number, but transversal intersections with very high condition number can also cause convergence problems.

[3] A specialization of a Bézier curve $b([a_1, a_2])$ is also a Bézier curve.

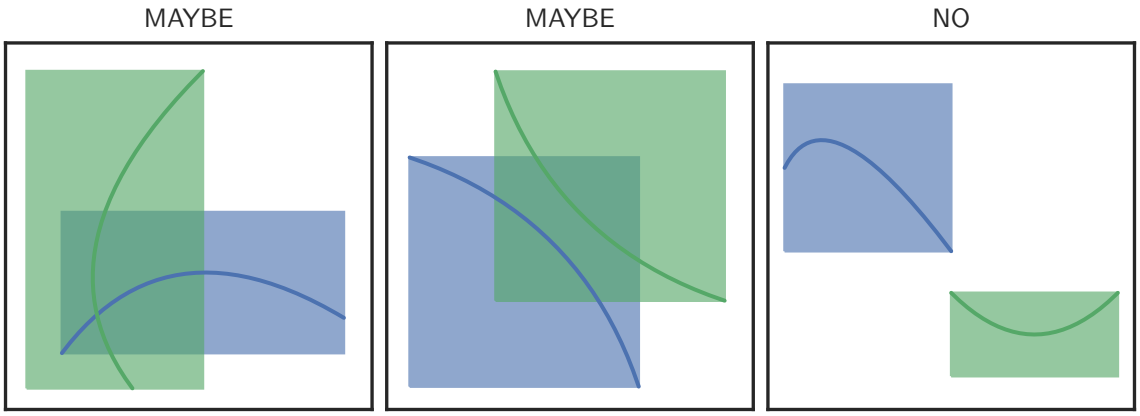

Figure 10: Bounding box intersection predicate. This is a cheap way to conclude that two curves don't intersect, though it inherently is susceptible to false positives.

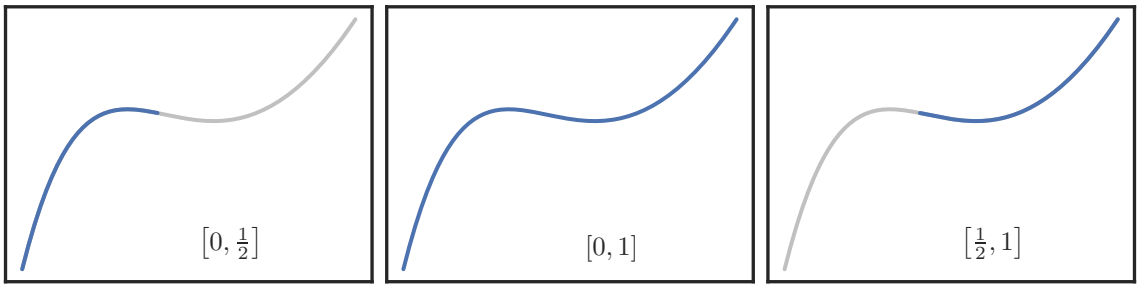

Figure 11: Bézier curve subdivision.

In the Bézier subdivision algorithm, we first check if the bounding boxes for the curves are disjoint (Figure 10). We use the bounding boxes rather than the convex hulls since they are easier to compute and the intersections of boxes are easier to check. If they are disjoint, the pair can be rejected. If not, each curve $\mathcal{C} = b([0, 1])$ is split into two halves by splitting the unit interval: $b\left(\left[0, \frac{1}{2}\right]\right)$ and $b\left(\left[\frac{1}{2}, 1\right]\right)$ (Figure 11).

As the subdivision continues, some pairs of curve segments may be kept around that won't lead to an intersection (Figure 12). Once the curve segments are close to linear within a given tolerance (Figure 13), the process terminates.

Once both curve segments are linear (to tolerance), the intersection is approximated by intersecting the lines connecting the endpoints of each curve segment. This approximation is used as a starting point for Newton's method, to find a root of $F(s, t) = b_0(s) - b_1(t)$. Since $b_0(s), b_1(t) \in \mathbf{R}^2$ we have Jacobian $J = \begin{bmatrix} b_0'(s) & -b_1'(t) \end{bmatrix}$. With these, Newton's method is

$$\begin{bmatrix} s_{n+1} & t_{n+1} \end{bmatrix}^T = \begin{bmatrix} s_n & t_n \end{bmatrix}^T - J_n^{-1} F_n. \tag{15}$$

This also gives an indication why convergence issues occur at non-transveral intersections: they are exactly the intersections where the Jacobian is singular.

3.2 Intersecting Bézier Triangles

The chief difficulty in intersecting two surfaces is intersecting their edges, which are Bézier curves. Though this is just a part of the overall algorithm, it proved to be the *most difficult* to implement ([Her17]). So the first part of the algorithm is to find all points where the edges intersect (Figure 14).

To determine the curve segments that bound the curved polygon region(s) (see Section 2.6 for more about curved polygons) of intersection, we not only need to keep track of the coordinates of intersection, we also

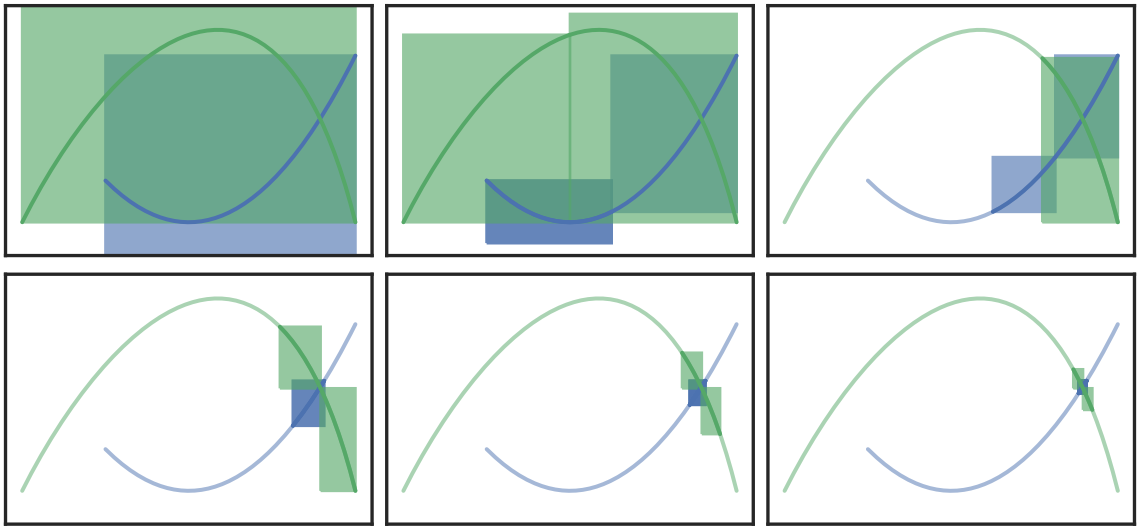

Figure 12: Bézier subdivision algorithm.

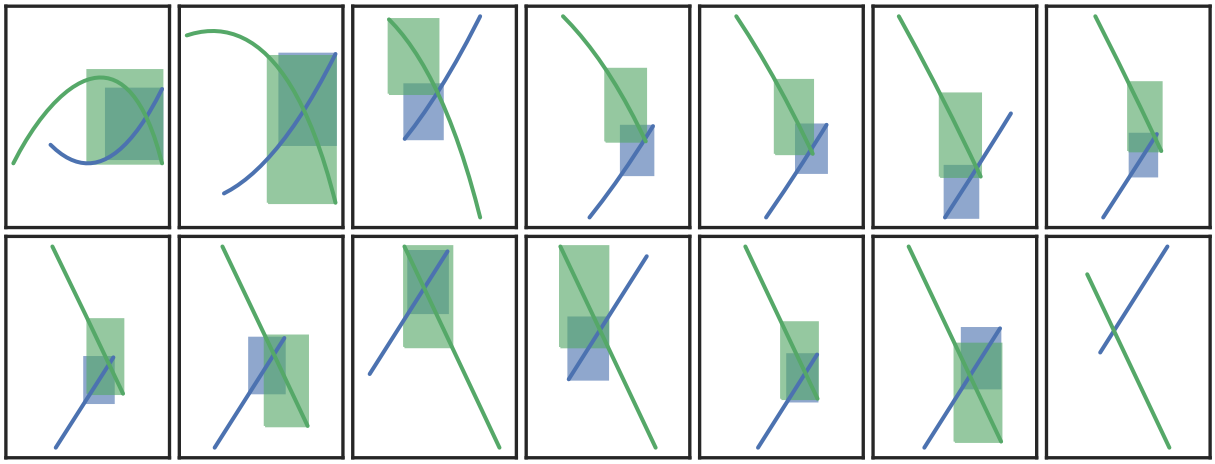

Figure 13: Subdividing until linear within tolerance.

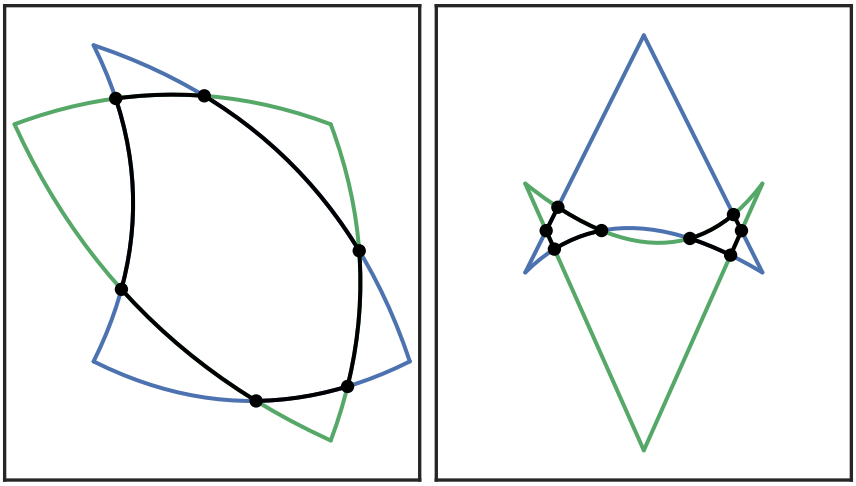

Figure 14: Edge intersections during Bézier triangle intersection.

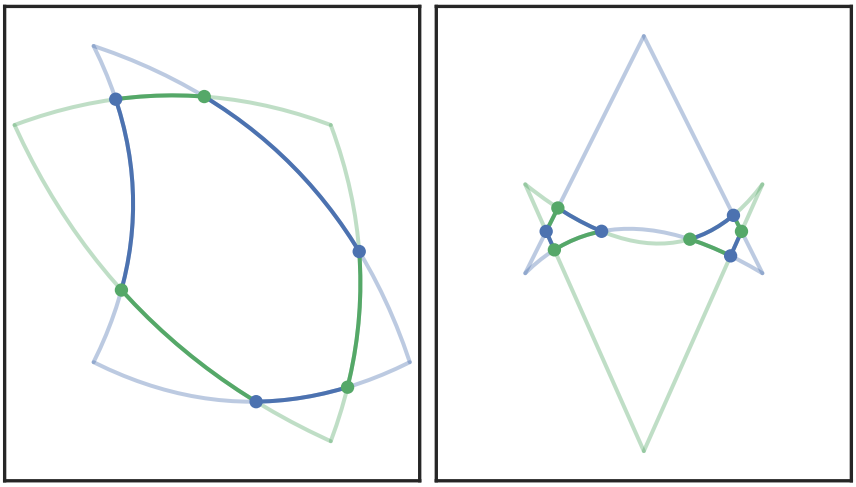

Figure 15: Classified intersections during Bézier triangle intersection.

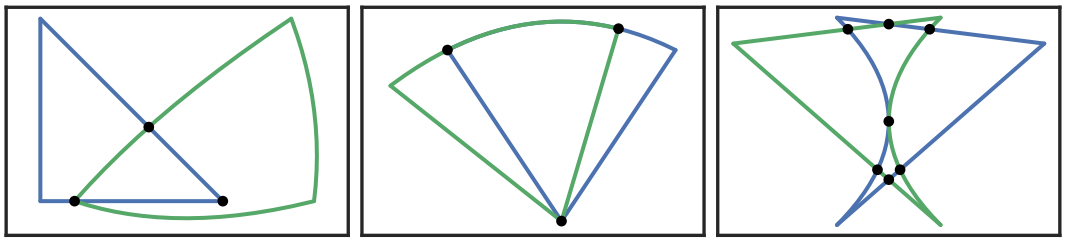

Figure 16: Bézier triangle intersection difficulties.

need to keep note of *which* edges the intersection occurred on and the parameters along each curve. With this information, we can classify each point of intersection according to which of the two curves forms the boundary of the curved polygon (Figure 15). Using the right-hand rule we can compare the tangent vectors on each curve to determine which one is on the interior.

This classification becomes more difficult when the curves are tangent at an intersection, when the intersection occurs at a corner of one of the surfaces or when two intersecting edges are coincident on the same algebraic curve (Figure 16).

In the case of tangency, the intersection is non-transversal, hence has infinite condition number. In the case of coincident curves, there are infinitely many intersections (along the segment when the curves coincide) so the subdivision process breaks down.

3.2.1 Example

Consider two Bézier surfaces (Figure 17)

$$b_0(s,t) = \begin{bmatrix} 8s \\ 8t \end{bmatrix} \qquad b_1(s,t) = \begin{bmatrix} 2(6s+t-1) \\ 2(8s^2+8st-8s+3t+2) \end{bmatrix} \tag{16}$$

In the *first step* we find all intersections of the edge curves

$$E_0(r) = \begin{bmatrix} 8r \\ 0 \end{bmatrix}, E_1(r) = \begin{bmatrix} 8(1-r) \\ 8r \end{bmatrix}, E_2(r) = \begin{bmatrix} 0 \\ 8(1-r) \end{bmatrix},$$
$$E_3(r) = \begin{bmatrix} 2(6r-1) \\ 4(2r-1)^2 \end{bmatrix}, E_4(r) = \begin{bmatrix} 10(1-r) \\ 2(3r+2) \end{bmatrix}, E_5(r) = \begin{bmatrix} -2r \\ 2(5-3r) \end{bmatrix}. \tag{17}$$

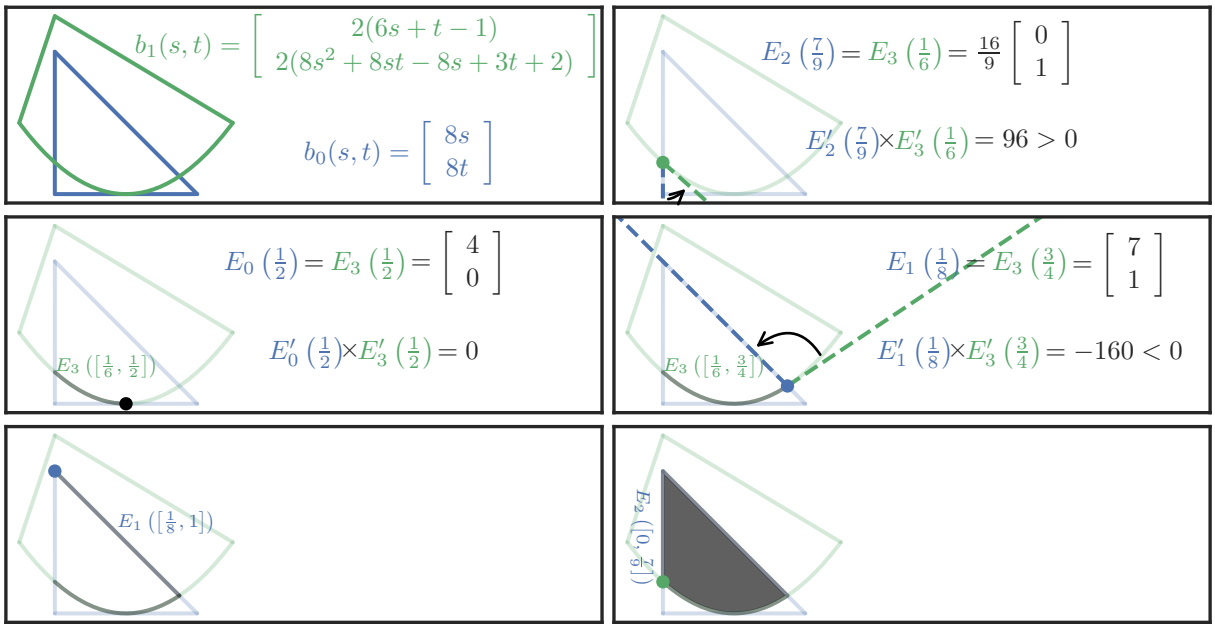

Figure 17: Surface Intersection Example

We find three intersections and we classify each of them by comparing the tangent vectors

$$I_1 : E_2\left(\frac{7}{9}\right) = E_3\left(\frac{1}{6}\right) = \frac{16}{9}\begin{bmatrix} 0 \\ 1 \end{bmatrix} \Longrightarrow E_2'\left(\frac{7}{9}\right) \times E_3'\left(\frac{1}{6}\right) = 96 \tag{18}$$

$$I_2 : E_0\left(\frac{1}{2}\right) = E_3\left(\frac{1}{2}\right) = \begin{bmatrix} 4 \\ 0 \end{bmatrix} \Longrightarrow E_0'\left(\frac{1}{2}\right) \times E_3'\left(\frac{1}{2}\right) = 0 \tag{19}$$

$$I_3 : E_1\left(\frac{1}{8}\right) = E_3\left(\frac{3}{4}\right) = \begin{bmatrix} 7 \\ 1 \end{bmatrix} \Longrightarrow E_1'\left(\frac{1}{8}\right) \times E_3'\left(\frac{3}{4}\right) = -160. \tag{20}$$

From here, we construct our curved polygon intersection by drawing from our list of intersections until none remain.

- First consider I_1. Since $E_2' \times E_3' > 0$ at this point, then we consider the curve E_3 to be *interior*.
- After classification, we move along E_3 until we encounter another intersection: I_2
- I_2 is a point of tangency since $E_0'\left(\frac{1}{2}\right) \times E_3'\left(\frac{1}{2}\right) = 0$. Since a tangency has no impact on the underlying intersection geometry, we ignore it and keep moving.
- Continuing to move along E_3, we encounter another intersection: I_3. Since $E_1' \times E_3' < 0$ at this point, we consider the curve E_1 to be *interior* at the intersection. Thus we stop moving along E_3 and we have our first curved segment: $E_3\left(\left[\frac{1}{6}, \frac{3}{4}\right]\right)$
- Finding no other intersections on E_1 we continue until the end of the edge. Now our (ordered) curved segments are:

$$E_3\left(\left[\frac{1}{6}, \frac{3}{4}\right]\right) \longrightarrow E_1\left(\left[\frac{1}{8}, 1\right]\right). \tag{21}$$

- Next we stay at the corner and switch to the next curve E_2, moving along that curve until we hit the next intersecton I_1. Now our (ordered) curved segments are:

$$E_3\left(\left[\frac{1}{6}, \frac{3}{4}\right]\right) \longrightarrow E_1\left(\left[\frac{1}{8}, 1\right]\right) \longrightarrow E_2\left(\left[0, \frac{7}{9}\right]\right). \tag{22}$$

  Since we are now back where we started (at I_1) the process stops

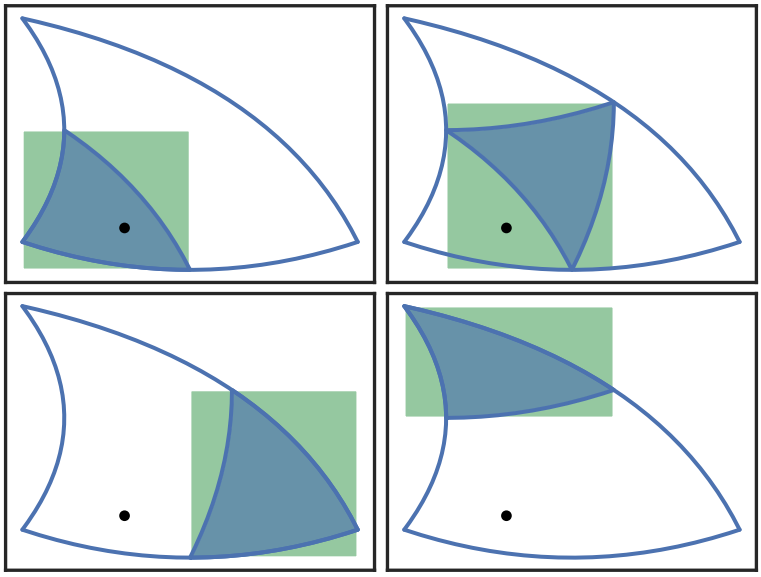

Figure 18: Checking for a point $\boldsymbol{p}$ in each of four subregions when subdividing a Bézier triangle.

We represent the boundary of the curved polygon as Bézier curves, so to complete the process we reparameterize ([Far01, Ch. 5.4]) each curve onto the relevant interval. For example, E_3 has control points $p_0 = \begin{bmatrix} -2 \\ 4 \end{bmatrix}$, $p_1 = \begin{bmatrix} 4 \\ -4 \end{bmatrix}$, $p_2 = \begin{bmatrix} 10 \\ 4 \end{bmatrix}$ and we reparameterize on $\alpha = \frac{1}{6}, \beta = \frac{3}{4}$ to control points

$$q_0 = E_3\left(\frac{1}{6}\right) = \frac{16}{9}\begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{23}$$

$$q_1 = (1-\alpha)\left[(1-\beta)p_0 + \beta p_1\right] + \alpha\left[(1-\beta)p_1 + \beta p_2\right] = \frac{1}{6}\begin{bmatrix} 21 \\ -8 \end{bmatrix} \tag{24}$$

$$q_2 = E_3\left(\frac{3}{4}\right) = \begin{bmatrix} 7 \\ 1 \end{bmatrix}. \tag{25}$$

3.3 Bézier Triangle Inverse

The problem of determining the parameters (s, t) given a point $\boldsymbol{p} = \begin{bmatrix} x & y \end{bmatrix}^T$ in a Bézier triangle can also be solved by using subdivision with a bounding box predicate and then Newton's method at the end.

For example, Figure 18 shows how regions of $\mathcal{U}$ can be discarded recursively until the suitable region for (s, t) has a sufficiently small area. At this point, we can apply Newton's method to the map $F(s, t) = b(s, t) - \boldsymbol{p}$. It's very helpful (for Newton's method) that $F : \mathbf{R}^2 \longrightarrow \mathbf{R}^2$ since the Jacobian will always be invertible when the Bézier triangle is valid. If $\boldsymbol{p} \in \mathbf{R}^3$ then the system would be underdetermined. Similarly, if $\boldsymbol{p} \in \mathbf{R}^2$ but $b(s)$ is a Bézier curve then the system would be overdetermined.

4 Solution Transfer

Placeholder.

5 Conclusion

This work has described a method for conservative interpolation between curved meshes. The transfer process conserves globally to machine precision since we can use exact quadratures for all integrals. The

primary source of error comes from solving the linear system with the mass matrix for the target mesh. This allows less restrictive usage of mesh adaptivity, which can make computations more efficient. Additionally, having a global transfer algorithm allows for remeshing to be done less frequently.

The algorithm breaks down into three core subproblems: Bézier triangle intersection, an advancing front for intersecting elements and integration on curved polygons. The inherently local nature of the advancing front allows the algorithm to be parallelized via domain decomposition with little data shared between processes. By restricting integration to the intersection of elements from the target and donor meshes, the algorithm can accurately transfer both continuous and discontinuous fields.

5.1 Future Work

As mentioned in the preceding chapters, there are several research directions possible to build upon the solution transfer algorithm. The usage of Green's theorem nicely extends to $\mathbf{R}^3$ via Stoke's theorem, but the Bézier triangle intersection algorithm is specific to $\mathbf{R}^2$. The equivalent Bézier tetrahedron intersection algorithm is significantly more challenging.

The restriction to shape functions from the global coordinates basis is a symptom of the method and not of the inherent problem. The pre-image basis has several appealing properties, for example this basis can be precomputed on $\mathcal{U}$. The problem of a valid tessellation of a curved polygon warrants more exploration. Such a tessellation algorithm would enable usage of the pre-image basis.

The usage of the global coordinates basis does have some benefits. In particular, the product of shape functions from different meshes is still a polynomial in $\mathbf{R}^2$. This means that we could compute the coefficients of $F = \phi_0\phi_1$ directly and use them to evaluate the antiderivatives H and V rather than using the fundamental theorem of calculus. Even if this did not save any computation, it may still be preferred over the FTC approach because it would remove the usage of quadrature points outside of the domain $\mathcal{P}$.

References

[Ber87] Marsha J. Berger. On Conservation at Grid Interfaces. *SIAM Journal on Numerical Analysis*, 24(5):967–984, Oct 1987.

[BR78] I. Babuška and W. C. Rheinboldt. Error Estimates for Adaptive Finite Element Computations. *SIAM Journal on Numerical Analysis*, 15(4):736–754, 1978.

[CDS99] Xiao-Chuan Cai, Maksymilian Dryja, and Marcus Sarkis. Overlapping Nonmatching Grid Mortar Element Methods for Elliptic Problems. *SIAM Journal on Numerical Analysis*, 36(2):581–606, Jan 1999.

[CH94] G. Chesshire and W. D. Henshaw. A Scheme for Conservative Interpolation on Overlapping Grids. *SIAM Journal on Scientific Computing*, 15(4):819–845, Jul 1994.

[CMOP04] David E. Cardoze, Gary L. Miller, Mark Olah, and Todd Phillips. A Bézier-Based Moving Mesh Framework for Simulation with Elastic Membranes. In *Proceedings of the 13th International Meshing Roundtable, IMR 2004, Williamsburg, Virginia, USA, September 19-22, 2004*, pages 71–80, 2004.

[DK87] John K. Dukowicz and John W. Kodis. Accurate Conservative Remapping (Rezoning) for Arbitrary Lagrangian-Eulerian Computations. *SIAM Journal on Scientific and Statistical Computing*, 8(3):305–321, May 1987.

[Duk84] John K Dukowicz. Conservative rezoning (remapping) for general quadrilateral meshes. *Journal of Computational Physics*, 54(3):411–424, Jun 1984.

[Far91] R.T. Farouki. On the stability of transformations between power and Bernstein polynomial forms. *Computer Aided Geometric Design*, 8(1):29–36, Feb 1991.

[Far01] Gerald Farin. *Curves and Surfaces for CAGD, Fifth Edition: A Practical Guide (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann, 2001.

[FM11] P.E. Farrell and J.R. Maddison. Conservative interpolation between volume meshes by local Galerkin projection. *Computer Methods in Applied Mechanics and Engineering*, 200(1-4):89–100, Jan 2011.

[FPP+09] P.E. Farrell, M.D. Piggott, C.C. Pain, G.J. Gorman, and C.R. Wilson. Conservative interpolation between unstructured meshes via supermesh construction. *Computer Methods in Applied Mechanics and Engineering*, 198(33-36):2632–2642, Jul 2009.

[GKS07] Rao Garimella, Milan Kucharik, and Mikhail Shashkov. An efficient linearity and bound preserving conservative interpolation (remapping) on polyhedral meshes. *Computers & Fluids*, 36(2):224–237, Feb 2007.

[HAC74] C.W Hirt, A.A Amsden, and J.L Cook. An arbitrary Lagrangian-Eulerian computing method for all flow speeds. *Journal of Computational Physics*, 14(3):227–253, Mar 1974.

[Her17] Danny Hermes. Helper for Bézier Curves, Triangles, and Higher Order Objects. *The Journal of Open Source Software*, 2(16):267, Aug 2017.

[IK04] Armin Iske and Martin Käser. Conservative semi-Lagrangian advection on adaptive unstructured meshes. *Numerical Methods for Partial Differential Equations*, 20(3):388–411, Feb 2004.

[JH04] Xiangmin Jiao and Michael T. Heath. Common-refinement-based data transfer between nonmatching meshes in multiphysics simulations. *International Journal for Numerical Methods in Engineering*, 61(14):2402–2427, 2004.

[JM09] Claes Johnson and Mathematics. *Numerical Solution of Partial Differential Equations by the Finite Element Method (Dover Books on Mathematics)*. Dover Publications, 2009.

[KLS98] Deok-Soo Kim, Soon-Woong Lee, and Hayong Shin. A cocktail algorithm for planar Bézier curve intersections. *Computer-Aided Design*, 30(13):1047–1051, Nov 1998.

[KS08] M. Kucharik and M. Shashkov. Extension of efficient, swept-integration-based conservative remapping method for meshes with changing connectivity. *International Journal for Numerical Methods in Fluids*, 56(8):1359–1365, 2008.

[MD92] Dinesh Manocha and James W. Demmel. Algorithms for Intersecting Parametric and Algebraic Curves. Technical Report UCB/CSD-92-698, EECS Department, University of California, Berkeley, Aug 1992.

[MM72] R. McLeod and A. R. Mitchell. The Construction of Basis Functions for Curved Elements in the Finite Element Method. *IMA Journal of Applied Mathematics*, 10(3):382–393, 1972.

[MS03] L.G. Margolin and Mikhail Shashkov. Second-order sign-preserving conservative interpolation (remapping) on general grids. *Journal of Computational Physics*, 184(1):266–298, Jan 2003.

[PBP09] P.-O. Persson, J. Bonet, and J. Peraire. Discontinuous Galerkin solution of the Navier–Stokes equations on deformable domains. *Computer Methods in Applied Mechanics and Engineering*, 198(17-20):1585–1595, Apr 2009.

[PUdOG01] C.C. Pain, A.P. Umpleby, C.R.E. de Oliveira, and A.J.H. Goddard. Tetrahedral mesh optimisation and adaptivity for steady-state and transient finite element calculations. *Computer Methods in Applied Mechanics and Engineering*, 190(29-30):3771–3796, Apr 2001.

[PVMZ87] J Peraire, M Vahdati, K Morgan, and O.C Zienkiewicz. Adaptive remeshing for compressible flow computations. *Journal of Computational Physics*, 72(2):449–466, Oct 1987.

[SN90] T.W. Sederberg and T. Nishita. Curve intersection using Bézier clipping. *Computer-Aided Design*, 22(9):538–549, Nov 1990.

[SP86] Thomas W Sederberg and Scott R Parry. Comparison of three curve intersection algorithms. *Computer-Aided Design*, 18(1):58–63, Jan 1986.

[WFA+13] Z.J. Wang, Krzysztof Fidkowski, Rémi Abgrall, Francesco Bassi, Doru Caraeni, Andrew Cary, Herman Deconinck, Ralf Hartmann, Koen Hillewaert, H.T. Huynh, Norbert Kroll, Georg May, Per-Olof Persson, Bram van Leer, and Miguel Visbal. High-order CFD methods: current status and perspective. *International Journal for Numerical Methods in Fluids*, 72(8):811–845, Jan 2013.

[Zlá73] Miloš Zlámal. Curved Elements in the Finite Element Method. I. *SIAM Journal on Numerical Analysis*, 10(1):229–240, Mar 1973.

[Zlá74] Miloš Zlámal. Curved Elements in the Finite Element Method. II. *SIAM Journal on Numerical Analysis*, 11(2):347–362, Apr 1974.

A Allowing Tessellation with Inverted Triangles

Lemma A.1. Consider three smooth curves b_0, b_1, b_2 that form a closed loop: $b_0(1) = b_1(0)$, $b_1(1) = b_2(0)$ and $b_2(1) = b_0(0)$. Take *any* smooth map $\varphi(s,t)$ on $\mathcal{U}$ that sends the edges to the three curves:

$$\varphi(r,0) = b_0(r), \quad \varphi(1-r,r) = b_1(r), \quad \varphi(0,1-r) = b_2(r) \quad \text{for } r \in [0,1]\,. \tag{26}$$

Then we must have

$$2\int_{\mathcal{U}} \det(D\varphi)\,[F\circ\varphi]\;dt\,ds = \oint_{b_0\cup b_1\cup b_2} H\,dy - V\,dx \tag{27}$$

for antiderivatives that satisfy $H_x = V_y = F$.

When $\det(D\varphi) > 0$, this is just the change of variables formula combined with Green's theorem.

Proof. Let $x(s,t)$ and $y(s,t)$ be the components of φ. Define

$$\Delta S = H(x,y)y_s - V(x,y)x_s \quad \text{and} \quad \Delta T = H(x,y)y_t - V(x,y)x_t. \tag{28}$$

On the unit triangle $\mathcal{U}$, Green's theorem gives

$$\int_{\mathcal{U}} [\partial_s \Delta T - \partial_t \Delta S]\;dV = \oint_{\partial\mathcal{U}} \Delta S\,ds + \Delta T\,dt. \tag{29}$$

The boundary $\partial\mathcal{U}$ splits into the bottom edge E_0, hypotenuse E_1 and left edge E_2.

Since

$$E_0 = \left\{ \begin{bmatrix} r \\ 0 \end{bmatrix} \mid r \in [0,1] \right\} \tag{30}$$

we take $\varphi(r,0) = b_0(r)$ hence

$$dx = x_s\,dr, dy = y_s\,dr \Longrightarrow H\,dx - V\,dy = \Delta S\,dr. \tag{31}$$

We also have $ds = dr$ and $dt = 0$ due to the parameterization, thus

$$\int_{E_0} \Delta S\,ds + \Delta T\,dt = \int_{r=0}^{r=1} \Delta S\,dr = \int_{b_0} H\,dx - V\,dy. \tag{32}$$

We can similarly verify that $\int_{E_j} \Delta S\,ds + \Delta T\,dt = \int_{b_j} H\,dx - V\,dy$ for the other two edges. Combining this with (29) we have

$$\int_{\mathcal{U}} [\partial_s \Delta T - \partial_t \Delta S]\;dV = \oint_{b_0\cup b_1\cup b_2} H\,dx - V\,dy. \tag{33}$$

To complete the proof, we need

$$\int_{\mathcal{U}} [\partial_s \Delta T - \partial_t \Delta S]\;dV = 2\int_{\mathcal{U}} \det(D\varphi)\,[F\circ\varphi]\;dV \tag{34}$$

but one can show directly that

$$\partial_s \Delta T - \partial_t \Delta S = 2\,(x_s y_t - x_t y_s)\,F(x,y) = 2\det(D\varphi)\,[F\circ\varphi]\,. \qquad \blacksquare$$